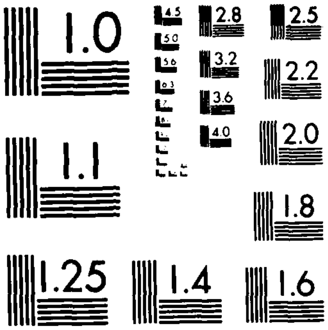
AD-A175 009   ADA COMPILER VALIDATION SUMMARY REPORT· SYSTEAM KG   1/1
SYSTEAM-GERMAN MOD S15 SIEMENS 7536(U)
INDUSTRIEANLAGEN-BETRIEBSGESELLSCHAFT M B H OTTOBRUNN
UNCLASSIFIED   (GERMANY F R)   06 JUN 86   F/G 9/2   NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

: (When Data Entered)

ENTATION PAGE

2. GOVT ACCESSION NO.	READ INSTRUCTIONS BEFORE COMPLETEING FORM 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: SYSTEAM KG SYSTEAM-GERMAN MoD S1.5 Siemens 7.536	5. TYPE OF REPORT & PERIOD COVERED June 1986 to June 1987 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* IABG m.b.h., Dept. SZT Einsteinstrasse 20 D 8012 Ottobrunn	8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION AND ADDRESS IABG M.B.H., Dept. SZT	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Programming Office United States Depar :ment of Defense Washington, D.C. ' 0301-3081	12. REPORT DATE 6 June 1986 13. NUMBER OF PAGES 42
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* IABG m.b.h., Dept. SZT Einsteinstrasse 20 D 8012 Ottobrunn	15. SECURITY CLASS *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20. If different from Report)*

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See Attached.

Ada* COMPILER VALIDATION SUMMARY REPORT:
SYSTEAM KG
SYSTEAM-German MoD S1.5
Siemens 7.536

Completion of On-Site Validation:
86-06-24

Prepared By:
IABG m.b.H., Dept SZT
Einsteinstrasse 20
D 8012 Ottobrunn

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

* Ada is a registered trademark of the United States Government (Ada Joint Program Office)

Ada* Compiler Validation Summary Report:

Compiler Name: SYSTEAM-German MoD S1.5

Host Computer	Target Computer
Siemens 7.536 under BS2000 V7.5	Siemens 7.536 under BS2000 V7.5

Testing Completed 86-06-24 Using ACVC 1.7

This report has been reviewed and approved:

Helmut Hummel

Ada Validation Facility
IABG m.b.H., Dept SZT
Dr. H. Hummel
IABG, Dept SZT
Einsteinstrasse
D 8012 Ottobrunn

John F Kramer

Ada Validation Office (AVO)
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria, VA

Virginia L. Castor

Ada Joint Program Office (AJPO)
Virginia L. Castor
Director
Washington, D.C.

* Ada is a registered trademark of the United States Government (Ada Joint Program Office)

EXECUTIVE SUMMARY

The Validation Summary Report presents the results and conclusions of testing performed on the SYSTEAM-German MoD S1.5 compiler. Standardized tests serve as input to an Ada compiler, producing result , which are evaluated by the validation team. This summary briefly states the highlights of the SYSTEAM-German MoD S1.5 validation.

On-site testing was performed 86-06-04 through 86-06-24 at D-8012 Ottobrunn under the auspices of the IABG m.b.H., Dept SZT (AVF), according to Ada Validation Office policies and procedures. The SYSTEAM-German MoD S1.5 is hosted on Siemens 7.536 operating under BS2000 V7.5. The suite of tests known as the Ada Compiler Validation Capability (ACVC), Version 1.7, was used. The ACVC is used to validate conformance of a compiler to ANSI/MIL-STD-1815A Ada. The purpose of testing is to ensure that a compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, or during execution.

The results of validation are summarized in the following table.

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	811	1118	17	9	21	2044
Failed	0	0	0	0	0	0	0
Inapplicable	0	13	202	0	2	2	219
Anomalous	0	0	0	0	0	0	0
Withdrawn	0	4	12	0	0	0	16
TOTAL	68	828	1332	17	11	23	2279

Tests found to contain errors were withdrawn from Version 1.7 of the Ada Compiler Validation Capability (ACVC). When validation was completed, the tests listed in Chapter 2.2 had been withdrawn.

Some tests demonstrate that language features are not supported by an implementation. For this implementation the tests determined the following.

- SHORT_INTEGER is not supported:

  B52004E-AB.DEP B55B09D-AB.DEP B86001CR-AB.DEP
  C34001D-B.DEP C55B07B-AB.DEP

- LONG_INTEGER is not supported:

  B52004D-AB.DEP B55B09C-AB.DEP B86001CS-AB.DEP
  C34001E-B.DEP C55B07A-AB.DEP

- SHORT_FLOAT is not supported:

  B86001CP-AB.DEP C34001F-B.DEP C35702A-AB.DEP

- LONG_FLOAT is not supported:

  B86001CQ-AB.DEP C34001G-B.DEP C35702B-AB.DEP

- Representation specifications for noncontiguous enumeration representations are allowed:

  C55B16A-AB.DEP

- No integer type other than INTEGER, SHORT_INTEGER, AND LONG_INTEGER is supported:

  B86001DT-AB.DEP

- The package SYSTEM is used by package TEXT_IO:

  C86001F-B.DEP

- The 'SIZE clause is supported:

  C87B62A-B.DEP

- The 'STORAGE_SIZE clause is supported:

  C87B62B.DEP

- The 'SMALL clause is supported:

  C87B62C-B.DEP

- Generic unit specifications and bodies can be compiled in separate compilations :

  CA1012A-B.DEP
  CA2009C-B.DEP
  CA200F-B.DEP

- Pragma INLINE is not supported for procedures:

  LA3004A-AB.ADA EA3004C-B.ADA CA3004E-B.ADA

- Pragma INLINE is not supported for functions:

  LA3004B-AB.ADA EA3004D-B.ADA CA3004F-B.ADA

- Mode IN_FILE is supported (for sequential I/O):

  CE2102D-B.ADA

- Mode OUT_FILE is supported (for sequential I/O):

  CE2102E-B.ADA

- Mode INOUT_FILE is supported (for direct I/O):

  CE2102F-B.ADA

- Mode RESET and DELETE are supported (for sequential and direct I/O):

  CE2102G-B.ADA

- Mode IN_FILE is supported (for direct I/O):

  CE2102I-B.ADA

- Mode OUT_FILE is supported (for direct I/O):

  CE2102J-B.ADA

- Dynamic creation and deletion of files are allowed:

  CE2106A-B.ADA CE3110A-B.DEP

- No more than one internal file can be associated with the same external file, except for reading:

  CE2107B-B.ADA CE2107C-B.ADA CE2111D-B.ADA
  CE3114B-B.ADA CE3111B-B.ADA CE3111C-B.ADA

- More than one internal file can be associated with the some external file for reading:

  CE2107A-B.ADA CE2107F-B.ADA CE3111A-B.ADA

- Instantiation of package SEQUENTIAL_IO with unconstrained array types is allowed:

  CE2201D-B.DEP

- Instantiation of package SEQUENTIAL_IO with unconstrained record types with discriminants is allowed:

  CE2201E-B.DEP

- Dynamic creation and resetting of files is supported:

  CE2210A-B.ADA

- Instantiation of package DIRECT_IO with unconstrained array types and unconstrained types with discriminants is supported:

  CE2401D-B.DEP

- An external file associated with more than one internal file cannot be reset:

  CE3115A-B.ADA

- Illegal filenames can exist:

  CE2102C-B.DEP

- Discriminant constraints are not allowed before full type declaration:

  C48006B-B.ADA   B74207A-B.ADA
  B37004A-B.ADA   BC3503A-B.ADA
  B38105B-AB.ADA

- Execution of library tests is discountinued after termination of the main program (see AI-00399):

  C94004A-B.ADA
  C94004B-B.ADA
  C94004C-B.ADA

ACVC Version 1.7 was present on-site on magnetic tape at D-8012 Ottobrunn. The tape was loaded, and all tests, except for the executable tests which make use of a floating point precision greater than SYSTEM.MAX_DIGITS, were compiled on Siemens 7.536. Class A, C, D, and E tests were executed on Siemens 7.536.

On completion of testing, all results were analyzed for failed Class A, C, D, or E programs, and all Class B and L compilation results were individually analyzed.

The ACVC, Version 1.7, contains 2279 tests of which 2044 were applicable to SYSTEAM-German MoD S1.5. No anomalies were found in the testing of this compiler. Testing demonstrated that all applicable tests were passed by this compiler and conformed to the Ada Standard. The AVF concluded that the results show acceptable compliance to ANSI/MIL-STD-1815A Ada.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Validation Summary Report describes how an Ada compiler conforms to the language standard. This report explains all technical terms used within and thoroughly reports the Ada Compiler Validation Capability (ACVC) test results. Ada compilers must be written according to the language specification as given in the ANSI/MIL-STD-1815A Ada. All implementation-defined features must be included for the compiler to conform to the Standard. Following the guidelines of the Standard ensures continuity between compilers. That is, the entire Standard must be implemented, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Standard, it must be understood that some differences do exist between implementations. ANSI/MIL-STD-1815A permits some implementation dependencies, e.g., the maximum length of identifiers, the maximum values of integer types, etc. These implementation-dependent features limit the portability of programs between compilers. Other differences between compilers are due to limitations imposed on a compiler by the operating system and by the hardware. All of these dependencies are given in the report.

Validation summary reports are written according to a standardized format. Compiler users can, therefore, more easily compare the reports from several compilers when selecting a compiler for a given task. The validation report can be completed mostly from the test results produced during validation testing. Additional testing information is given at the end of the report and states problems and details which are unique for a specific compiler. The format of the validation report limits variance between reports, enhances readability of the report, and accelerates report readiness.

1.1 Purpose of this Validation Summary Report

The Validation Summary Report documents the results of the testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To identify any language constructs supported by the translator that do not conform to the Ada Standard

- To identify any unsupported language constructs required by the Ada Standard

- To describe the implementation-dependent behavior allowed by the Ada Standard

Testing of this compiler was conducted by IABG m.b.H., Dept SZT according to policies and procedures established by the Ada Validation Office (AVO). Testing was conducted from 86-06-04 through 86-06-24 at D-8012 Ottobrunn.

1.2 Use of this Validation Summary Report

Consistent with the national laws of the originating country, the Ada Validation Office may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. no 552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that any statement or statements set forth in this report are accurate or complete, or that the subject compiler has no nonconformances to the Ada Standard other than those presented. This report is not intended for the purpose of publicizing the findings summarized herein.

Questions regarding this report or the validation tests should be directed to:

Ada Validation Office
Institute for Defense Analyses
1801 N. Beauregard
Alexandria VA 22311

and to:

IABG m.b.H., Dept SZT
Einsteinstrasse
D 8012 Ottobrunn

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, Feb 1983

2. Ada Validation Organization - Policies and Procedures, Mitre Corporation, Jun 1982

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., Dec 1984.

1.4 DEFINITION OF TERMS

Anomaly
A test result that, given pre-validation analysis, is not expected during formal validation but is judged allowable under the circumstances.

ACVC
The Ada Compiler Validation Capability. A set of programs that evaluates the conformance of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.

Ada Standard
ANSI/MIL-STD-1815A, February 1983.

Applicant
The agency requesting validation.

AVF
The IABG m.b.H., Dept SZT. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.

AVO
The Ada Validation Office. In the context of this report, the AVO is responsible for setting policies and procedures for compiler validations.

Compiler
A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test
A test for which the compiler generates a result that demonstrates nonconformance to the Ada Standard.

Host
The computer on which the compiler resides.

Inapplicable test
A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test
A test for which a compiler generates the expected result.

Target
The computer for which a compiler generates code.

Test
A program that evaluates the conformance of a compiler to a language specification. In the context of this report, the term is used to designate a single ACVC test. The text of a program may be the text of one or more compilations.

Withdrawn test — A test that has an invalid test objective, fails to meet its test objective, or contains illegal use of the language.

1.5 Configuration

The candidate compilation system for this validation was tested under the configuration:

Compiler: SYSTEAM-German MoD S1.5

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computer:

Machine(s):	Siemens 7.536
Operating System:	BS2000 V7.5
Memory Size:	4 MB
Disk System:	Siemens Disk 3470

Target Computer:

Machine(s):	Siemens 7.536
Operating System:	BS2000 V7.5
Memory Size:	4 MB
Disk System:	Siemens Disk 3470

CHAPTER 2

TEST RESULTS

2.1 ACVC Test Classes

Conformance to ANSI/MIL-STD-1815A is measured using the Ada Compiler Validation Capability (ACVC). The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. Legal programs are compiled and executed while illegal programs are just compiled. Support packages are used to report the results of the legal programs. A compiler must correctly process each of the tests in the suite and demonstrate conformance to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Tests that are found to contain errors are withdrawn from the ACVC. The results of validation testing are summarized in the following table:

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	811	1118	17	9	21	2044
Failed	0	0	0	0	0	0	0
Inapplicable	0	13	202	0	2	2	219
Anomalous	0	0	0	0	0	0	0
Withdrawn	0	4	12	0	0	0	16
TOTAL	68	828	1332	17	11	23	2279

A total of 2093 tests were processed during this validation attempt. 16 withdrawn tests in Version 1.7 were not processed, nor were 170 Class C tests that were inapplicable because they use floating point types having digits that exceed the maximum value for the implementation. All other tests were processed. In addition, 7 tests (class C) for the report package were processed and passed.

Some conventions are followed in the ACVC to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic 55 character set, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values. The values used for this validation are listed in Appendix B.

2.1.1 Class A Tests

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a message indicating that it has passed. If a Class A test cannot be compiled and executed because of its size, then the test is split into a set of smaller subtests that can be processed. No splits were required for class A tests.

The following table shows that all applicable Class A tests passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	15	9	0	5	2	12	13	3	0	0	0	9	68
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	0	0	0	0	0	0	0	0	0	0
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	15	9	0	5	2	12	13	3	0	0	0	9	68

2.1.2 Class B Tests

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined manually to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler. If one or more errors are not detected, then a version of the test is created that contains only the undetected errors. The resulting "split" is compiled and examined. The splitting process continues until all errors are detected by the compiler. Splits were required for 3 tests:

B22003A B23004A B97101E

The following table shows that all applicable Class B tests passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	39	84	86	109	73	66	46	87	36	8	159	18	811
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	2	0	4	0	1	5	0	0	0	1	0	13
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	1	0	0	0	1	0	1	0	1	0	4
TOTAL	39	86	87	113	73	67	52	87	37	8	161	18	828

2.1.3 Class C Tests

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASS/FAIL message indicating the result when it is executed. If a Class C test cannot be compiled because it exceeds the compiler's capacity, then the test is split into smaller subtests until all are compiled and executed. No splits were required for class C tests:

The following table shows that all applicable Class C tests passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	39	133	215	117	82	18	96	108	42	20	56	192	1118
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	21	76	87	2	0	0	1	1	2	0	0	12	202
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	1	3	0	0	0	0	2	5	0	0	1	12
TOTAL	60	210	305	119	82	18	97	111	49	20	56	205	1332

2.1.4 Class D Tests

Class D tests check the compilation and execution capacities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for the number of identifiers permitted in a compilation, the number of units in a library, the number of nested loops in a subprogram body, and so on, a compiler may refuse to compile a Class D test. Each Class D test is self-checking and produces a PASS/FAIL message indicating the result when it is executed. If a Class D test fails to compile because the capacity of the compiler is exceeded, then the test is classified as inapplicable.

The following table shows that all applicable Class D tests passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	1	0	4	9	3	0	0	0	0	0	0	0	17
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	0	0	0	0	0	0	0	0	0	0
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	1	0	4	9	3	0	0	0	0	0	0	0	17

Capacities measured by the Class D tests are detailed in section 2.4, IMPLEMENTATION CHARACTERISTICS.

2.1.5 Class E Tests

Class E tests provide information about the compiler in those areas in which the Ada Standard permits implementations to differ. Each Class E test is executable and produces messages that indicate how the Ada Standard is interpreted. However, in some cases the Ada Standard permits a compiler to detect a condition either at compile time or at execution time, and thus a Class E test may correctly fail to execute. A Class E test is passed if it fails to compile and appropriate error messages are issued, or if it executes properly and produces a message that it has passed. If a Class E test cannot be compiled and executed because of its size, then the test is split into a set of smaller subtests that can be processed. No splits were required for class E tests:

The following table shows that all applicable Class E tests passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	1	3	2	1	1	0	0	0	0	0	0	1	9
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	0	0	0	0	0	2	0	0	0	2
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	1	3	2	1	1	0	0	0	2	0	0	1	11

Information obtained from the Class E tests is detailed in section 2.4, IMPLEMENTATION CHARACTERISTICS.

2.1.6 Class L Tests

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time and the test does not execute.

The following table shows that all applicable Class L tests passed:

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	0	0	0	0	0	0	0	0	21	0	0	0	21
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	0	0	0	0	0	0	0	0	2	0	0	0	2
Anomalous	0	0	0	0	0	0	0	0	0	0	0	0	0
Withdrawn	0	0	0	0	0	0	0	0	0	0	0	0	0
TOTAL	0	0	0	0	0	0	0	0	23	0	0	0	23

2.1.7 Support Units

Three packages support the self-checking features of Class C tests: REPORT, CHECK_FILE, and VAR_STRINGS. The REPORT package provides the mechanism by which executable tests report results. It also provides a set of identity functions that are used to defeat some compiler optimization strategies to cause computations to be made by the target computer instead of the by the compiler on the host computer. The CHECK_FILE package is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The VAR_STRINGS package defines types and subprograms for manipulating varying-length character strings. The operation of these three packages is checked by a set of executable tests. These tests produce messages that are examined manually to verify that the packages are operating correctly. If these packages are not operating correctly, then validation is not attempted.

An applicant is permitted to substitute the body of package REPORT with an equivalent one if for some reason the original version provided by the ACVC cannot be executed on the target computer. Package REPORT was modified for this validation in order to print an identifying message as well as date and time of execution.

All support package specifications and bodies were compiled and were demonstrated to be operating correctly.

2.2 Withdrawn Tests

Some tests are withdrawn from the ACVC because they dc not conform to the Ada Standard. When testing was performed, the following 16 tests had been withdrawn for the reasons indicated:

C35904A: The elaboration of subtype declarations SFX3 & SFX4 may raise NUMERIC_ERROR vs. CONSTRAINT_ERROR.

C41404A: The values of 'LAST and 'LENGTH in the "if" statements from line 74 to the end of the test are incorrect

C48008A: This test requires that the evaluation of default initial values not occur if an exception is raised by an allocator. However, the LMC has ruled that such a requirement is incorrect (AI-00397).

B4A010C: The object_declaration in line 18 follows a subprogram body of the same declarative part.

C4A014A: The number declarations in lines 19-22 are not correct, because conversions are not static.

B83A06B: The Ada Standard 8.3(17) and AI-00330 permit the label LAB_ENUMERAL of line 80 to be considered a homograph of the enumeration literal in line 25.

C92005A. At line 40, "/=" for type PACK.BIG_INT is not visible without a "use" clause for package PACK.

C940ACA This test assumes that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program; however, such an execution order is not required by the Ada Standard, so the test is erroneous.

CA1003B: This test requires all of the legal compilation units of a file containing some illegal units to be compiled and executed. But according to AI-00255 such a file may be rejected as a whole.

BA2001E: The Ada Standard 10.2(5) states that "simple names of all subunits that have the same ancestor library unit must be distinct identifiers." This test checks for the above condition when stubs are declared; but it is not clear that the check must be made then, as opposed to when the subunit is compiled.

CA3005A..D (4 tests): There exists no valid elaboration order for these tests.

BC3204C: The file BC3204C4 should contain the body for BC3204C0 --as indicated in line 25 of BC3204C3M.

CE2107E TEMP_HAS_NAME must be given an initial value of TRUE.

2.3 Inapplicable Tests

N/A-Tests	count	reason
C24113L .. Y	14	Value of $Digits exceeds SYSTEM.MAX_DIGITS (170 tests)
C35705L .. Y	14	
C35706L .. Y	14	
C35707L .. Y	14	
C35708L .. Y	14	
C35802L .. Y	14	
C45241L .. Y	14	
C45321L .. Y	14	
C45421L .. Y	14	
C45424L .. Y	14	
C45521L .. Z	15	
C45621L .. Z	15	
C24113E .. K	7	Source lines longer than 80 characters
C34001D .. G	4	The implementation does not support SHORT_INTEGER, LONG_INTEGER, other INTEGER types, SHORT_FLOAT, or LONG_FLOAT
C35702A .. B	2	
B52004D .. E	2	
B55B09C .. D	2	
C55B07A .. B	2	
B86001CP.. S	4	
C86001F	1	package SYSTEM is used by package TEXT_IO
CA3004E .. F	2	pragma INLINE is not supported
LA3004A .. B	2	

N/A-Tests	count	reason
CE2107B .. D	3	one internal file can be associated with more than one external file only for reading
CE2110B	1	
CE2111D	1	
CE2111H	1	
CE3111B .. E	4	
CE3114B	1	
CE3115A	1	
B86001DT	1	the only predefined numeric types are integer and float
96005B	1	no duration'base values outside type duration exist
EA3004C .. D	2	pragma inline has no effect
C48006B	1	discriminant constraints are not allowed before full type declaration
B37004A	1	
B38105B	1	
B74207A	1	
BC3503A	1	

total count 219

2.4 Implementation Characteristics

One of the purposes of validation is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, inapplicable tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

. Non-graphic characters.

Non-graphic characters are defined in the ASCII character set but are not permitted in Ada programs, even within character strings. The compiler correctly recognizes these characters as illegal in Ada compilations. The characters are not printed in the output listing but are contained in the protocol files on disk.

. Capacities.

The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, procedures nested to 17 levels, and 723 variables.

. Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly.

. Predefined types.

This implementation does not support numeric types other than INTEGER and FLOAT.

. Based literals.

An implementation is allowed to reject a based literal with value exceeding SYSTEM.MAX_INT during compilation or it may raise NUMERIC_ERROR during execution. This compiler raises NUMERIC_ERROR during execution.

. Array types.

An implementation is allowed to raise NUMERIC_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. When an array type is declared with an index range exceeding INTEGER values and with a component that is a null BOOLEAN array, this compiler raises NUMERIC_ERROR when the type is declared.

When an array type is declared with an index range exceeding SYSTEM.MAX_INT values and with a component that is a null BOOLEAN array, this compiler raises NUMERIC_ERROR when on object of this type is declared.

A packed BOOLEAN array of length INTEGER'LAST+3 raises NUMERIC_ERROR when the array objects are declared. A packed two-dimensional BOOLEAN array with INTEGER'LAST+3 components raises NUMERIC_ERROR when the array objects are declared.

A null array with one dimension of length exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared.

In assigning one-dimensional array types, the entire expression is evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is comptaible with the target's subtype. In assigning two-dimensional array types, the entire expression is not evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning record types with discriminants, the entire expression is evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype.

- Discriminated types.

An incompletely declared type with discriminants may not be used in an access type definition and constrained either there or in later subtype indications.

- Aggregates.

When evaluating the choices of a multi-dimensional aggregate all choices are evaluated before checking against the index type.

When evaluating an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds.

- Functions.

The declaration of a parameterless function with the same profile as an enumeration literal in the same immediate scope is allowed by the implementation.

- Representation clauses.

  'SMALL length clauses are supported.

  Enumeration representation clauses are supported.

- Generics.

  When given a separately compiled generic declaration, some illegal instantiations, and a body, compiler rejects the body because of the instantiations.

- Package CALENDAR.

  TIME_OF and SPLIT are inverses when SECONDS is a non-model number.

- Pragmas.

  Pragma INLINE is not supported for procedures. It is not supported for functions.

- Input/output.

  Package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. Package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. A form parameter is needed in the case of DIRECT_IO and unconstrained array types.

  More than one internal file can be associated with each external file for sequential I/O, direct I/O, and text I/O for reading only. An external file associated with more than one internal file cannot be deleted.

  An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and can be created in IN_FILE mode.

  Dynamic creation and resetting of a sequential file is allowed.

  Temporary sequential files are given a name. Temporary direct files are given a name. Temporary text files given names are deleted when they are closed.

  Temporary text files have a name. Temporary textfiles are deleted when closed.

CHAPTER 3

Compiler Anomalies and Nonconformances

3.1 Anomalies

An anomaly is a test result that, given the pre-validation analysis, was not expected during formal validation but which is judged allowable by the AVF and the AVO under the circumstances of the validation. No anomalies were detected in this validation attempt.

3.2 Nonconformances

Any discrepancy between expected test results and actual test results is considered to be a nonconformance. No nonconformances were detected in this validation attempt.

CHAPTER 4

ADDITIONAL TESTING INFORMATION

4.1 Pre-Validation

Prior to validation, a set of test results for ACVC 1.7 produced by the SYSTEAM-German MoD S1.5 compiler was submitted to IABG m.b.H., Dept SZT by the applicant for pre-validation review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests.

4.2 Test Site

Tests were compiled and executed at IABG in Ottobrunn on the AVF's Siemens 7.536 computer.

4.3 Test Tape Information

The original tapes containing ACVC version 1.7 as received from the AVO were read by the validation team. The withdrawn tests were deleted (except for one whose result was ignored) and the tests which make use of implementation dependent parameters were customized.

4.4 Testing Logistics

Once all tests had been loaded to disk, processing was begun using command scripts provided by the AVF. Samples of the text of these scripts are given in Appendix C.

The compiler supports various options that control its operation. The options used for testing are evident from the example in Appendix C. The B-tests were run with the additional compiler option LIST=>OV, which produces a full compilation protocol including error messages.

First a project library REPLIB was prepared to contain the report package. Then a number of batch jobs were initiated, one at a time, to process the tests. Each job for executable tests created a new project library with REPLIB as parent library. Test results were written to system files in concatenated form. These files were written on tape in backup format and archived.

4.5 Testing Duration

The ACVC has not been designed for use in measuring compiler performance. However, information about the length of time needed to test the compiler may characterize compiler performance in processing a large number of programs.

Testing started at 86-06-04 and was completed on 86-06-24. The machine idled for about 100 hours because of power disturbances.

CHAPTER 5

SUMMARY AND CONCLUSIONS

The IABG m.b.H., Dept SZT identified 2093 of the 2279 tests in Version 1.7 of the Ada Compiler Validation Capability to be processed during the validation of SYSTEAM-German MoD S1.5. Because of test errors, 16 tests were withdrawn. 219 tests were not applicable, 170 of them because they use floating point types having digits that exceed the maximum value for that implementation. The remaining 2044 processed tests were passed by the compiler.

The IABG m.b.H., Dept SZT concludes that these results demonstrate acceptable conformance to the Ada Standard.

APPENDIX A

COMPLIANCE STATEMENT

The only allowed implementation dependencies correspond to implementation-dependent pragmas and attributes, to certain machine-dependent conventions as mentioned in Chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the SYSTEAM-German MoD S1.5 are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A).

(1) Implementation-Dependent Pragmas

INTERFACE

Takes ASSEMBLER ,subprogram_name as argument(s).
This pragma is allowed at the place of declarative items.
This pragma specifies that an object module generated
by the system assembler is supplied for the subprogram.

SUPPRESS-ALL

Takes no argument. This pragma is allowed at
the place of the start of a compilation.
This pragma specifies that all checks which may
raise CONSTRAINT_ERROR at runtime are suppressed.

(2) Implementation-Dependent Attributes

HEAP_ADDRESS The value of this attribute is of type ADDRESS.

(3) Package SYSTEM

The specification for package SYSTEM is

```
package SYSTEM is

type ADDRESS is private;
type NAME is (siemens_bs2000);

SYSTEM_NAME  : constant NAME := siemens_bs2000;
STORAGE_UNIT : constant := 8;
MEMORY_SIZE  : constant := 5*2:1:E20; -- 5MB

--  System-Dependent Named Numbers:

MIN_INT      : constant := -2_147_483_648;
MAX_INT      : constant :=  2_147_483_647;
MAX_DIGITS   : constant := 15;
MAX_MANTISSA : constant := 51;
FINE_DELTA   : constant := 2:1.0:E-30;
TICK         : constant := 2:1.0:E-14;

--  Other System-Dependent Declarations

subtype PRIORITY is INTEGER range  0 .. 255;

type    UNIVERSAL_INTEGER is range
        MIN_INT .. MAX_INT;
subtype EXTERNAL_ADDRESS is STRING;
type    ADDRESS_RANGE is range
        0 .. 2:1:E24-1;
function CONVERT_ADDRESS (ADDR:  EXTERNAL_ADDRESS)
                          return ADDRESS;
function CONVERT_ADDRESS (ADDR:  ADDRESS)
                          return EXTERNAL_ADDRESS;
function CONVERT_ADDRESS (ADDR:  ADDRESS_RANGE)
                          return ADDRESS;
function CONVERT_ADDRESS (ADDR:  ADDRESS)
                          return ADDRESS_RANGE;
function "+"             (ADDR:  ADDRESS;
                          OFFSET:INTEGER)
                          return ADDRESS;
private
   type ADDRESS is access BOOLEAN;
end SYSTEM;
```

(4) Representation Clause Restrictions

Representation clauses specify how the types of the language are to be mapped onto the underlying machine. The following are restrictions on representation clauses.

for t'size use static_expression

For integer, enumeration and fixed point type t, 16, 24 and 32 are allowed as value of static_expression depending on the range and the 'small of t. For access types t, only 32 is allowed as value of static_expression. For floating point types t, only 64 is allowed as value of static_expression.

For record and array types, the value of static_expression must match the size computed by the compiler. This means that the type mapping for records and arrays cannot be influenced by a 'size rep.spec.

for 'small use static_expression

For the value of the static_expression only a power of two, i.e. 2.0**k for some integer k, is allowed.

for access_type' storage_size use expression.
for task_type storage_size use expression

There is no restriction concerning the value of expression.

Address Clause

Is implemented for objects only.

(5) Conventions

There are no implementation-generated names denoting implementation-dependent components.

(6) Address Clauses

The following are conventions that define the interpretation of expressions that appear in address clauses, including those for interrupts.

The object starts at the given address. For objects accessed by a descriptor, the descriptor starts at the given address.

An object for which an address specification is given must not require an initialization.

(7) Unchecked Conversions

The following are restrictions on unchecked conversions, including those depending on the respective sizes of objects of the source and target.

If TARGET'SIZE > SOURCE'SIZE results will be unpredictable.

(8) Input-Output Packages

The following are implementation-dependent characteristics of the input-output packages.

SEQUENTIAL IO Package

```
type FILE_TYPE is limited private;  --  integer

procedure CREATE ( ... );   function MODE        ( ... );
procedure OPEN   ( ... );   function NAME        ( ... );
procedure CLOSE  ( ... );   function FORM        ( ... );
procedure DELETE ( ... );   function IS_OPEN     ( ... );
procedure RESET  ( ... );   function END_OF_FILE ( ... );
procedure READ   ( ... );
procedure WRITE  ( ... );
```

DIRECT IO Package

```
     type COUNT is range 0 .. 2_147_483_647;
```

TEXT IO Package

```
     type COUNT is range 0 .. 2_147_483_647;

     subtype FIELD is INTEGER range 0 .. 255;
```

LOW LEVEL IO

```
type DEVICE_TYPE is (NULL_DEVICE);
type DATA_TYPE   is record null; end record;
procedure SEND_CONTROL    (DEVICE : DEVICE_TYPE;
                              DATA : in out DATA_TYPE);
procedure RECEIVE_CONTROL (DEVICE : DEVICE_TYPE;
                              DATA : in out DATA_TYPE);
```

(9) Package STANDARD

```
type INTEGER is range -2_147_483_648 .. 2_147_483_647;

type FLOAT is digits 15
              range     -16:0.FFFF_FFFF_FFFF_FF:E+63

                     ..  16:0.FFFF_FFFF_FFFF_FF:E+63;

type DURATION is delta 2:1.0:E-14
                 range -131_072.0 ..
                        131_071.999_938_964_843_75;
```

(10) File Names

File names make use of the following conventions and restrictions.

They must be BS2000-file-names, max. 54 characters long,upper case letters. At most 15 user-defined files may be open at a time.

(11) Other Characteristics

The maximum source program line length is 80.

The program library may contain at most 2_000 units. One compilation unit may import at most 63 units directly.

APPENDIX B

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

$MAX_IN_LEN Maximum input line length permitted by the implementation.	80
$BIG_ID1 Identifier of size MAX_IN_LEN with varying last character.	string (1 .. 80) := (1 .. 79 => 'A', 80 => '1')
$BIG_ID2 Identifier of size MAX_IN_LEN with varying last character.	string (1 .. 80) := (1 .. 79 => 'A', 80 => '2')
$BIG_ID3 Identifier of size MAX_IN_LEN with varying middle character.	string (1 .. 80) := (1 .. 40 => 'A', 41 => '1', 42 .. 80 => 'A')
$BIG_ID4 Identifier of size MAX_IN_LEN with varying middle character.	string (1 .. 80) := (1 .. 40 => 'A', 41 => '2', 42 .. 80 => 'A')

$NEG_BASED_INT A based integer literal whose highest order non-zero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16:FFFFFFFE:
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is MAX_IN_LEN characters long.	string (1 .. 80) := (1 .. 77 => '0', 78 .. 80 => "298")
$BIG_REAL_LIT A real literal that can be either of floating or fixed point type, has value 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long.	string (1 .. 80) := (1 .. 74 => '0', 75 .. 80 => "69.0E1")
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstu vwxyz!$%?@[\]^`{}~"
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)
$BLANKS Blanks of length MAX_IN_LEN - 20	string (1 .. 60) := (1..60 => ' ')
$MAX_DIGITS Maximum digits supported for floating point types.	15
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2147483647
$LESS_THAN_DURATION	- 0.0

A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	0.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST.	- 200_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST.	200_000.0
$COUNT_LAST Value of COUNT'LAST in TEXT_IO package.	2147483647
$FIELD_LAST Value of FIELD'LAST in TEXT_IO package.	255
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long.	abc! def.dat
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	abc*def.dat
$ILLEGAL_EXTERNAL_FILE_NAME1 Illegal external file name.	x$!yz.dat
$ILLEGAL_EXTERNAL_FILE_NAME2 Illegal external file names.	string (1 .. 60) := (1 .. 60 => 'A')

APPENDIX C

COMMAND SCRIPTS

```
/.#Z1#Z1 LOGON
/DO ¤SYSTEAM1.ADA.DELETELIB,LIBRARY=Z
/DO ¤SYSTEAM1.ADA.CREATELIB,LIBRARY=Z,PARENT=REPLIB
/WHEN ,ON=(1)
/DO ¤SYSTEAM1.ADA.COMPILE,
/¤COPY.ZZ.CZ1101A-AB.ADA,LIBRARY=Z
/DO ¤SYSTEAM1.ADA.LINK,CZ1101A,ACVC.CZ1101A,LIBRARY=Z
/SYSFILE SYSOUT=(ACVCRES.CZ,EXTEND)
/EXEC ACVC.CZ1101A
/ERASE ACVC.CZ1101A
/STEP
/SYSFILE SYSOUT=(PRIMARY)
/WHEN ,ON=(1)
/DO ¤SYSTEAM1.ADA.COMPILE,
/¤COPY.ZZ.CZ1102A-AB.ADA,LIBRARY=Z
/DO ¤SYSTEAM1.ADA.LINK,CZ1102A,ACVC.CZ1102A,LIBRARY=Z
/SYSFILE SYSOUT=(ACVCRES.CZ,EXTEND)
/EXEC ACVC.CZ1102A
/ERASE ACVC.CZ1102A
/STEP
/SYSFILE SYSOUT=(PRIMARY)
/WHEN ,ON=(1)
/DO ¤SYSTEAM1.ADA.COMPILE,
/¤COPY.ZZ.CZ1103A-B.ADA,LIBRARY=Z
/DO ¤SYSTEAM1.ADA.LINK,CZ1103A,ACVC.CZ1103A,LIBRARY=Z
/SYSFILE SYSOUT=(ACVCRES.CZ,EXTEND)
/EXEC ACVC.CZ1103A
/ERASE ACVC.CZ1103A
/STEP
/SYSFILE SYSOUT=(PRIMARY)
/WHEN ,ON=(1)
/DO ¤SYSTEAM1.ADA.COMPILE,
/¤COPY.ZZ.CZ1201A-AB.ADA,LIBRARY=Z
/DO ¤SYSTEAM1.ADA.LINK,CZ1201A,ACVC.CZ1201A,LIBRARY=Z
/SYSFILE SYSOUT=(ACVCRES.CZ,EXTEND)
/EXEC ACVC.CZ1201A
/ERASE ACVC.CZ1201A
/STEP
/SYSFILE SYSOUT=(PRIMARY)
/WHEN ,ON=(1)
/DO ¤SYSTEAM1.ADA.COMPILE,
/¤COPY.ZZ.CZ1201B-AB.ADA,LIBRARY=Z
/DO ¤SYSTEAM1.ADA.LINK,CZ1201B,ACVC.CZ1201B,LIBRARY=Z
/SYSFILE SYSOUT=(ACVCRES.CZ,EXTEND)
/EXEC ACVC.CZ1201B
/ERASE ACVC.CZ1201B
/STEP
/SYSFILE SYSOUT=(PRIMARY)
/WHEN ,ON=(1)
/DO ¤SYSTEAM1.ADA.COMPILE,
/¤COPY.ZZ.CZ1201C-AB.ADA,LIBRARY=Z
/DO ¤SYSTEAM1.ADA.LINK,CZ1201C,ACVC.CZ1201C,LIBRARY=Z
/SYSFILE SYSOUT=(ACVCRES.CZ,EXTEND)
/EXEC ACVC.CZ1201C
/ERASE ACVC.CZ1201C
/STEP
/SYSFILE SYSOUT=(PRIMARY)
/WHEN ,ON=(1)
/DO ¤SYSTEAM1.ADA.COMPILE,
/¤COPY.ZZ.CZ1201D-AB.ADA,LIBRARY=Z
/DO ¤SYSTEAM1.ADA.LINK,CZ1201D,ACVC.CZ1201D,LIBRARY=Z
/SYSFILE SYSOUT=(ACVCRES.CZ,EXTEND)
/EXEC ACVC.CZ1201D
/ERASE ACVC.CZ1201D
/STEP
/SYSFILE SYSOUT=(PRIMARY)
/LOGOFF
```

END

1-87

DTIC